Cartas de processamento de informações 39 (1991) 343-350
Holanda do Norte

27 de setembro de 1991

A abstração de ordenação causal e uma maneira simples de implementá-la

Michel Rayrai

IRt,$A, Beaulieu Campus, F-35042 Rennes Cedex, França

Outros Schippers

Escola Politécnica F•'d••e!c, Departamento de InJomiatiqve, C ff *-1015 Lausanne, França*

Sam Toweg

Departamento de Ciência da Computação, Cornell University, Três, *NT 94853, ttSA*

Comunicado por L. Kott Recebido
em 11 de maio de 1990
Revisado em 3 de abril de **1991**

resumo

Raynal, M., A. Schiper e S. Toueg, A abstração de ordem causal e uma maneira simples de implementá-la, Information .°roc•.ssing * •.tters 39 (1991) 343-350.

O controle em sistemas distribuídos é introduzido principalmente para reduzir o não determinismo. Este não determinismo deve-se, por um lado, à execução assíncrona dos processos localizados nos vários sites do sistema e, por outro lado, à natureza assíncrona dos canais de comunicação. Para limitar o assincronismo devido aos canais de comunicação, uma nova relação de ordenação de mensagens, conhecida como ordenação causal, foi introduzida por Birman e Joseph. Depois de dar alguns exemplos dessa ordenação causal, propomos um algoritmo simples para implementá-la. Este algoritmo é baseado na numeração da sequência de mensagens. Uma prova da correção do algoritmo também é fornecida.

Palavras-chave: Computação distribuída, algariii'ur• distribuído, eausai or3.•âog, n•-twork produ!

1. Controlando o não detenninismo

Controlar o não determinismo é uma das tarefas essenciais em sistemas computacionais. Em sistemas centralizados o não determinismo é causado pelas interrupções. Em sistemas distribuídos o não terminismo também se deve à execução assíncrona dos processos paralelos juntamente com a natureza assíncrona dos canais de comunicação . Além disso, tais sistemas também podem ser afetados por ocorrências não determinísticas de falhas de componentes.

Para reduzir o assincronismo dos processos e resolver os problemas relacionados (por exemplo, problema de alocação de recursos), geralmente se introduz a sincronização. mecanismos [14,12]. Para reduzir o assincronismo dos canais de comunicação é possível construir sincronizadores de rede que forçam processos e canais a progredir em passos sincronizados [1,7]. Isso corresponde a definir uma máquina virtual distribuída na qual o comportamento indesejável devido ao não terminismo foi suprimido. O mesmo vale para os protocolos

efetivamente para mascarar canais não confiáveis: os protocolos de comunicação definem uma abstração de "canal confiável" que é implementada sobre um canal não confiável. O protocolo de bit alternativo [2] ou o protocolo de Stenning [16] são alguns exemplos: ambos constroem um canal sem perda, duplicação ou deseqüenciamento de mensagem em cima de um canal que pode perder ou duplicar mensagens, ou ainda, no caso do protocolo de Stenning , que pode desequenciá-los. A abstração confiável do canal, assim como os mecanismos de sincronização, elimina os comportamentos indesejáveis.

Neste artigo, consideramos um sistema distribuído confiável e estamos preocupados com a realização de um esquema de comunicação que elimine um determinado comportamento indesejável. Mais precisamente, estamos preocupados com a abstração chamada "ordenação causal", conforme proposto pelo sistema Isis [3]. O artigo está organizado da seguinte maneira. Na Seção 2, definimos ordenação causal e mostramos sua utilidade. Na Seção 3 mostramos uma maneira fácil e natural de implementar a ordenação causal (que difere da implementação de Isis), e damos na Seção 4 uma prova de sua correção.

2. Ordenação causal

2.L *Dej'iniiion*

A ordenação causal de eventos em um sistema distribuído (um evento correspondente à emussão de uma mensagem, a recepção de uma mensagem ou uma ação interna) é baseada na conhecida relação "aconteceu antes", notada —• [9] . Se *A* e *B* são dois eventos, então *A* —• *B* se e somente se uma das seguintes condições for verdadeira: *(i) A* e *B* são dois eventos ocorrendo no mesmo local, *A* ocorre antes de *B-, (ii) A* é a emissão de uma mensagem., e *B* corresponde à recepção da mesma mensagem; *(iii)* existe um evento *C* tal que *A* —• *n"* e C' —• *B.*

A ordenação causal diz respeito à ordem na qual duas mensagens ñfl £tnd M tais que SEND(Mt) —• SEND(ñf2), são entregues. Vamos introduzir a notação **DELIVERY(3f)** para indicar o evento correspondente à entrega de 3f

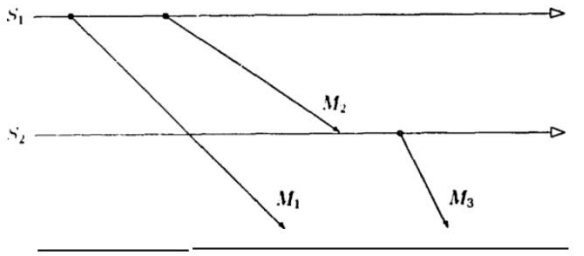

Fig. 1. A ordenação causal é respeitada se *M* for entregue antes de *Mz.*

ao seu processo de destino, e considere dois eventos **SEND(3f.,)** e SEND(f2) A ordem causal é respeitada se: se

SEND(ñf,) —• **SEND(ñf,),** e M o , *M* tem mesmo destino, então **D ELIVERY(arquivo) ENTREGA (ñf2)**

Em outras palavras, no caso de uma relação "aconteceu antes" entre duas emissões para um mesmo destino , existe uma relação "aconteceu antes" entre a entrega das mensagens. Por exemplo, na Fig. 1, a ordem causal é respeitada se a mensagem ñf, for entregue no site S3 antes da mensagem M_3.

Ao enviar mensagens usando protocolos de transporte tradicionais, a ordem causal pode ser claramente violada. Implementar ordenação causal significa adicionar um protocolo ao sistema original de forma que a ordenação causal nunca seja violada no sistema equipado com o protocolo sobreposto. Para evitar confusão entre os dois sistemas, usaremos a seguir consistentemente a expressão mensagem "recepção" para denotar uma mensagem recebida pelo sistema original, e mensagem "entrega" para denotar uma mensagem recebida pelo sistema equipado com o protocolo de ordenação causal. Uma analogia pode ser feita aqui entre canais FIrO e um sistema implementando ordenação causal. Um canal **FIFO** reduz o não determinismo de uma comunicação ponto a ponto. Um sistema implementando ordenação causal globalmente reduz a indeterminação das comunicações.

2.2. Exemplos da utilidade da ordenação causal

Gerenciamento de dados replicados

Considere um dado N replicado em vários locais. A fim de assegurar a consistência mútua de tii• vari-

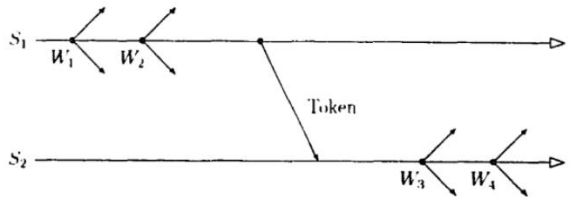

Fig. 2. Gestão de dados replicados.

ou cópias ml , x „ ... , as atualizações dessas cópias devem ocorrer na mesma ordem. Em particular, isso introduz a necessidade de exclusão mútua nas atualizações feitas por dois sites diferentes. Isso pode ser resolvido usando um token; quando de posse do token, um site pode atualizar sua cópia e transmitir a atualização para todos os outros sites que tenham uma cópia de A (Fig. 2). A ordenação causal garante que todos os sites recebam todas as atualizações na mesma ordem (IN, wi-fi não será entregue antes de lt„ para *i* <y), o que garante a consistência mútua das cópias [4,8]. O token garante a ordenação total do updateIs on e a ordenação causal garante que todas as cópias *x,* sejam atualizadas nesta mesma ordem.

Observando um sistema distribuído

A ordenação causal também pode ser usada para fornecer observações consistentes de um sistema distribuído. Considere vários sites monitorando eventos que ocorrem localmente e suponha que todas essas informações sejam de interesse de um sistema operacional de site observador. Se as informações de monitoramento coletadas pelos sites forem enviadas ao OS respeitando a ordem causal, as informações serão recebidas em uma origem coerente (ou seja, em uma ordem que não viole a ordem de precedência dos eventos). Por exemplo, na Fig. 1, o observador *S* recebe mensagens de sondagem M e depois Of„, ou seja, em uma ordem de observação consistente.

Alocação de recursos

Considere o problema de exclusão mútua ou, mais geralmente, um problema de alocação de recursos. Isso pode ser resolvido por um alocador de recursos residente em um site, que recebe solicitações de alocação em uma fila **FIFO .** Com esse esquema, os pedidos são atendidos na ordem de recebimento. Isso pode, no entanto, ser considerado insatisfatório em alguns casos: pode ser desejável honrar os pedidos não pela ordem de recepção, mas pela ordem de emissão, ou seja, se dois A; deve e *R* são tais que R, —+ A 2, então solicita ser honrado antes de fi Este é novamente um problema de ordenação causal (considere na Fig. 1 que *My* carrega a solicitação A e 3f3 carrega a solicitação A 2). Observe que este exemplo, no caso particular de exclusão mútua, foi usado por Lamport em seu artigo introduzindo relógios lógicos [9]. A solução de Lamport foi ordenar totalmente as solicitações, o que não é necessário se as mensagens forem ordenadas causalmente.

3. **Implementação de** ordenação causal

3.I. Trabalho relatado

A implementação de Isis da ordenação causal [3} é direta: uma mensagem carrega consigo toda a história das comunicações que a precederam. Considere novamente a Fig. 1. A mensagem M enviada por S2 para S carrega a seguinte i*.formação sobre seu passado: mensagem *My,* com a informação de que foi enviada para S3, e mensagem 3f Com a informação de que foi enviada para S2. Ao receber ñf3, o Site S3 também recebe uma cópia de *M :* se ainda não foi entregue, *M* será entregue naquele momento, ou seja, antes de *M .* Para evitar o crescimento ilimitado das informações adicionadas a uma mensagem, deve ser adicionado um mecanismo para informar as mensagens que foram recebidas e entregues: essas mensagens não precisam mais ser enviadas. Este mecanismo é, no entanto, bastante complexo. De qualquer forma, o tamanho do pneu das informações adicionadas às mensagens é ilim

Outra implementação de ordenação causal é dado em [15]. Nesta implementação, o controle na formação (e não nas mensagens como no Isis) é adicionado às mensagens. Esta informação de controlo permite ao local de destino de uma mensagem ñf saber se existem mensagens que têm de ser entregues antes de If, de forma a respeitar a ordem causal: se for o caso, ñf não é imediatamente entregue. A informação de controle é composta por um número limitado de pares (local de destino, vetor tempo), onde tempo" é um tempo que define vetor uma ordenação parcial dos ex'ents de um sistema distribuído [5,11]. A implementação proposta neste artigo tem algumas semelhanças com esta implementação. Não usamos tempos vetoriais e, portanto, esse algoritmo é muito mais fácil de entender.

3.2. Uma implementação simples

Considere que temos um sistema não causal original (isto é, onde a ordem causal pode ser violada) NC. Nosso objetivo é definir um protocolo no sistema NC para construir um novo sistema C no qual a ordenação causal nunca seja violada. Para implementar o protocolo, damos a cada site em informações de controle NC representando a percepção do site do estado do sistema (mais precisamente, a percepção do site das comunicações). Esta informação permitirá que cada site decida quando uma mensagem recebida pode ser entregue ao sistema C. Além disso, as informações adicionadas a cada mensagem permitirão que o site receptor no NC, quando uma mensagem for entregue, atualize sua percepção do estado do sistema.

Informações locais de um site Cada site no sistema NC gerencia as duas variáveis seguintes (onde ii é igual ao r:número de sites no sistema):

DELIV: array[1..n] de inteiro;

(' inicialmente DELIV[i] = 0 para todo i *)

SENT: array[1..n, 1..n] de inteiro;

(' inicialmente SENT[i, y] = 0 para todo i, *)

Usaremos a notação **DELIV,** e SENT, para nos referirmos às variáveis gerenciadas pelo site *S,.* Então no site *S„ a* variável DELIV,[y] representa o número de mensagens enviadas de S, e entregues a *S,,* a variável SENT,[k, *I* representa o conhecimento de *S,* do número de mensagens enviadas (mas não necessariamente entregue) de S para *St* (mais precisamente S , foi informado que pelo menos SENT,[k, I] mesias z'ere enviadas de *Sk* para *St).*

Para ilustrar essas definições, considere a Fig. 3 e, mais precisamente, SENT;[3, 1] que é igual a 1, significando que o site y sabe que uma mensagem (aqui fill) foi enviada do site *k* —- 3 para o site *i* = 1. Como veremos, esta informação foi enviada junto com ñf2.

Comportamento de um site O comportamento de um site *S* é expresso pelas duas regras a seguir, que regem a emissão e a recepção de uma mensagem no sistema NC.

[“i) Emissão de uma mensagem M de S, para S :

enviar(Se, ENVIADO,) para *S ;*

ENVIADO,[i, y] := ENVIADO, *[i,* y] + 1;

Assim, *M* é enviado junto com uma cópia de SENT„ ou seja, com o site *S,'S* visualização do estado do sistema.

('u) Recepção de M, ST f) *enviado de S para S„ onde* STD *representa a informação de controle* “SENT” *transportada pela mensagem M* (ver Fig. 3):

espera(para todo *k,* DELIV, *[k]* STD[ñ, *i]);*

entrega de *M* ao sistema C;

DELIV,[y] := DELIV,[y] + 1;

ENVIADO,[J, f '= ENVIADO,[y, *i]* + 1;

para todo *k, 1:* SENT,[k, I]

= max(ENVIADO,[k, *I],* STD[k, f]);

Assim, uma mensagem ñf enviada por fi, pode ser entregue a *S,* somente se para todo *k,* DELIV,[k] ST$ *k, i},* que expressa que todas as mensagens causalmente precedentes, cuja existência é revelada por STD, foram entregue. Considere, por exemplo, a mensagem *M* na Fig. 3. STp,{3, 1] = 1, o que significa que *M* pode ser entregue ao site 1 quando DELIV,[3] 1. Isso é verdade assim que ñf4 é entregue. Observe neste exemplo particular que DELIV [3] > STD,[3, 1] quando ñf4 é recebido no site 1. Esta situação se deve à existência de 3f3, cuja emissão não precede causalmente a emissão de *M :* SEND(ñf3) —• ENVIAR(ñf4). Os dois eventos **SEND(ñf3)** e SEND(ñf4) são considerados con atual.

Depois que uma mensagem é entregue, a percepção do site sobre o estado do sistema é atualizada de maneira óbvia.

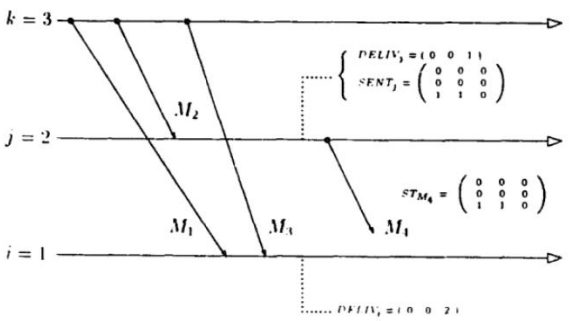

Fig. 3. Ilustração das estruturas de dados.

4. Prova de correção

A correção do algoritmo acima será provada nas duas etapas usuais; segurança e vivacidade. No entanto, antes de desenvolver essas provas, precisamos de alguns resultados intermediários que serão dados na Seção 4.1. Em seguida, provaremos na Seção 4.2 a segurança de nosso protocolo, o que corresponde a provar que a ordenação causal nunca é violada. Finalmente na Seção 4.3 provaremos a vivacidade do algoritmo, que corresponde a provar que toda mensagem será eventualmente entregue. Com relação ao sistema NC subjacente no qual a ordenação causal é implementada, precisamos assumir que todas as mensagens enviadas são recebidas, ou seja, nenhuma mensagem é perdida no NC e o sistema NC está ativo.

4.1. Resultados preliminares

A fim de provar a segurança e vivacidade do nosso protocolo, precisamos de alguns resultados preliminares.

Proposição 1. *Considere uma mensagem M senf por S para S; tal que* STD[k, *i]* > 0 *(incluindo k —- j), e seja* ST» !k, *i —— x. Então Mpa, a x-ésima mensagem enviada por Sk para S„ é tal que* SEND(Mf,) —• SEND(If).

Prova. (i) *k —— j.* De STp{j, i] = z e do protocolo de emissão, SENT,(y, *i]* = z no momento em que *M* é enviado. Além disso, segue diretamente do protocolo que SENT, (y, *i* nunca é incrementado ao receber uma mensagem por *S;.* Assim, no momento em que ñf é enviado, o protocolo de emissão foi executado *x* vezes por S, ou seja , *M* foi enviado antes de *M,* e SEND(3f,) —• SEND(M).

(ii) *k ¥ j.* A prova consistirá na construção de uma cadeia causal SEND(3f,) —• —• SEND(If).
Seja Nj a primeira mensagem enviada por *S;* tal que IT *g [k, i]* -- *x* (Fig. 4), ou seja, SEND,[k, *i]* —— *x* quando A, foi enviado (A, pode ser 3f). Assim, antes das emissões de A„ St deve ter recebido uma mensagem‹ige *My* • de *S* com STD *,[k, i - x* (consulte a última etapa do protocolo de entrega) e SEND(ñf,)

ENVIAR(3f).

Porque STD *,[k, i* - STD[k, *i),* a mesma construção aplicável a If pode ser aplicada a *My ,* e assim por diante. Cada iteração desta construção introduz um novo site *St(s)* diferente de todos os anteriores

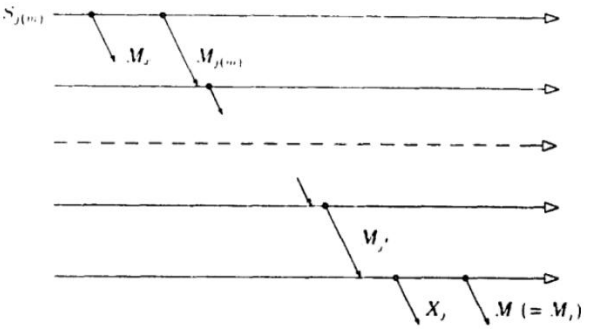

Fig. **4. Construção da** cadeia causal SEND(Mf,) —• •-- —• SEND(if).

sites , pois — • define uma ordem [9]: se - *St, , Sj,!2* for4r s1 s2, teríamos um ciclo na cadeia causal construída.

Como o número de sítios é finito, finalmente terminamos com S„„) = *S ,* 'It•› enviado por *S (q;,* e *STMG(k , i —— x,* que foi tratado em (i). Assim, a cadeia causal SEND(ñf,)

SEND(fj,p- -,) —• —• **SEND(ñf,)** —• SEND(M) foi criado. O

Os dois lemas a seguir são resultados intermediários necessários para provar a Proposição 4.

Lema 2. Seja *M a x-ésima mensagem enviada por St para S, x* > 0), *e considere uma mensagem M tal que* SEND(3fg) —• SEND(ñf). *Então* STD[j, i] > *x.*

Prova (esboço). Do protocolo de emissão, segue imediatamente que STD,[i, y] = z — 1. A partir disso, o lema pode ser facilmente comprovado considerando o protocolo de emissão e recepção aplicado a qualquer conexão de cadeia causal_• SF.ND(M ,) Em SEND(ñf). O

Lema *3.* *Considere uma mensagem M e /* ef STD[J, *i* —- *x. Então M+ a mensagem {x* + 1)sf *enviada por S, para S, é tal que S*'END(3fg+;) —• SEND(ñf).

Prova. Suponha que SEND(3fy+ t) —• ey SEND(*M*).
Lemma 2 temos ST3f[I , *i]* >px + 1, o que mostra a contradição. O

Proposição *4.* *Considere M enviado por S para S„ e outro sítio Sz (includf* Ztg *k —— j). Então* STD[k, f] *é igual ao número de mensagens N. enviadas por S; para S, tal que* SEND(3f') —• SEND(ñf).

Ptmf. Seja m o número de mensagens *M'* enviadas por *Sz* para S, tal que SEND(Cf) —• SEND(ñf). e seja STE *k, i - x.* (i)

Pela Proposição 1, o . a mensagem enviada de *S* para *S,* é tal que SEND(ñf,) —• SENDtCf). Assim m z. (ii) Considere M de S, para *S,.* , a (.x + l}ª mensagem enviada Pelo Lema 3. SENO(ñf,qt)-• SEND(3f). Assim *m < x.*

De (i) e (ü) concluímos que ni = z. O

Corolário 5. *Considere um arquivo de mensagens se i front St para S„ e uma mensagem M; tal que* SENO(ñf;) —• SENO(ñf). T7iex STp,[y, i] < STD,[y. eu].

Prmf. Pela Proposição 4. STD,(y, *i}* é igual ao número de mensagens *M'* enviadas de *St* para S, tal que SENO(ñf') —• SENO(ñf,). Mas temos SEND(ñf') - + SEND(3ft) —• SENDt fill). Assim, porque âf também é enviado de S, para *S„ pela* Proposição 4, STp,{y, i] é pelo menos igual a STD,[j, i] + 1. Isso permite concluir que ST„„[j, *i]* < STp,{y, i]. O

Proposição 6. *Considere um itiessage M enviado de St* para S,. *IJ nem M nem qualquer* mensagem *M' tal que* SENO(M) —• SENDtM') *foi entregue a S„ 'depois* DELIV,{j] *y* ST„{j, i].

Prova. Temos apenas que considerar as mensagens 3f' enviadas por *St,* apenas a entrega dessas mensagens pode aumentar o DELIV,[j]. Por Proposição. 4, STp[y, *i]* é igual ao número de mensagens ñf" enviadas por S, a fi, tais que **SFV *If*** *M") —•* SEND(3f). Por hipótese, somente essas mensagens ñf" poderiam ter sido entregues a S,. Assim DELIV,{y] STD[y,i]. O

Finalmente, a Proposição 7 será usada apenas para provar a vivacidade do nosso protocolo. Por esta razão podemos, na Proposição 7, assumir que nosso protocolo é seguro, ou seja, se SEND(3f,) —• SEND(ñf2) anJ *MM*,são enviados para o mesmo destino, então ñf2 Não pode ser entregue antes de *My* (safety será provado na Seção 4.il).

Proposição 7. *Suponha que nosso protcCol* seja *seguro, e considere .W„ a x-ésima mensagem enviada de S para S,* (z > 0). *Então se* DELIV,[J] < x. *M não foi entregue a S,.*

Prova. DELIV,[y] é incrementado cada vez que uma mensagem de *Si* é entregue. Assim, se DELIV,[y] < *x,* menos que x mensagens de *Si* foram entregues *a S,.* Como o protocolo é considerado seguro, no máximo (x - 1) as primeiras mensagens de *Si* poderiam ter sido entregues a *S,.* Assim, ñf, não foi entregue a *S,.* O

4.?. Segurança

A segurança corresponde a provar que a ordem causal nunca é violada. isto é, se SEND(ñf,) —+ SEN O (ñf2) e arquivo *M;* são enviados para o mesmo destino. então >2 •não pode ser entregue antes de *M .* Isso pode ser reformulado da seguinte maneira. Considere uma mensagem *M* enviada de S, para S„ e seja *M* denotar a enésima mensagem entregue a Então, para todo ii, enquanto ñf não for entregue, SENO(ñf) —• SENO(ñf) não pode ser verdadeiro. Esta fórmula é adaptada para uma prova por indução.

(i) *Base stef*• (ri = 1). Vamos supor que ñf;, a primeira mensagem entregue a *S„ é* tal que SENO(ñf) —• SEND(ñf,). e mostre que chegamos a uma contradição. Pela Proposição 4, como *M* é enviado de *S* para S„ STp,{y, i] 1. Inicialmente DñLiY,[j} = 0, então quando *M* é recebido por *S„ DEL.iV,*[y] < STD, [y, *i]:* a condição de entrega (Seção 3.2) não é satisfeita, ou seja, *M* não pode ser entregue, o que mostra a contradição.

(ii) *Etapa de indução.* Por hipótese de indução, enquanto ñf não for entregue, para todo pn, SEND(ñf) —• SENO(ñf) não pode ser verdadeiro. Vamos provar que o mesmo vale para ii + 1. Suponhamos novamente que SEND(3f) —• SEND(ñf ,) e mostre a contradição.

Antes de ñf, + t ser entregue por *S;,* temos pela Proposição 6: DELIV,[y] STD[y, *i].* Ainda mais pelo Corolário 5 temos STp[y, i] < STp„„[y, *i],* ou seja, DELIV,[y] < STp„„[y, *i]* antes da recepção de ñf + , por *S,.* Assim, a condição de entrega não é satisfeita (Seção 3.2), ou seja, ñf, +, não pode ser entregue. Isso novamente mostra a contradição .

4.3. hiueness

Como já dito, provaremos aqui que toda mensagem é eventualmente entregue. Considere duas mensagens If', *M“* e defina 3f' < *M“* iff SEND(ñf') —• SEND(ñf"). Suponhamos que uma mensagem seja entregue assim que for recebida e sua condição de entrega se torne verdadeira. Portanto, uma mensagem não pode ser entregue se não for recebida ou se for recebida e sua condição de entrega for falsa.
Considere então um site S, e todas as mensagens enviadas para ele que não podem ser entregues. Usando a relação <, seja *M* uma das menores dessas mensagens (pode haver mais de uma dessas mensagens, pois < é uma ordem parcial). Esta mensagem 3f foi enviada por algum *St.* Quando recebida, se 3f, não puder ser entregue, temos (consulte a Seção 3.2):

3k tal que DELIV,[k] < STE,(k, i].

Seja STE,(k, *i]* = x, e considere *M„ a* x-ésima mensagem enviada de *Sz* para *S,.* De DELIV,[*£*] < x e da Proposição 7 deduzimos que M, não foi entregue a S,. Além disso , pela Proposição 1, temos SEND(3f,) —• SEND(ñf,), ou seja, *Mg* < *M .*
Isso significa que ihat ñf não é uma das menores mensagens que não podem ser entregues a *S,* o que mostra a contradição. Segue-se que toda mensagem recebida por S eventualmente será entregue.

5. CONCLUSÃO

A ordenação causal das mensagens reduz o não determinismo decorrente da asiricronia dos canais de comunicação. Alguns exemplos foram dados para ilustrar a utilidade dessa noção de ordenação. Um algoritmo simples implementando esta ne ção, baseado na contagem de mensagens, foi proposto. Com essa implementação, o site de destino de uma mensagem sabe, ao receber uma mensagem, se ela pode ser entregue imediatamente ou se as mensagens precedentes ainda estão em andamento. Em contraste com a implementação potencialmente ilimitada do Ísis, a informação adicionada às mensagens para garantir a ordenação causal é aqui uma matriz nx *n ,* onde ii é o número de sites (cada entrada da matriz, no entanto, não é limitada). º•. mesmo limite é obtido em (15], mas a implementação dada é mais complicada. Em contraste, a implementação proposta é simplesmente baseada na contagem das mensagens emitidas. Em certo sentido, esta implementação estende o conceito de números de sequência usados em redes [16]. De forma mais geral, as técnicas de contagem são usuais em muitas soluções para problemas de sincronização, desde semáforos em sistemas centralizados (para resolver problemas de competição ou sincronização), até contadores em sistemas distribuídos usados para obter informações globais consistentes (por exemplo, rescisão distribuída [6,10]) ou para garantir decisões globais coerentes (por exemplo, exclusão mútua distribuída [13]).

Reconhecimento

Os autores gostariam de agradecer aos revisores por suas sugestões que ajudaram a melhorar a clareza do artigo.

Referências

[1] B. Awerbuch, Complexidade da sincronização de redes, J. *ACM* 32 (4) (1985) 804-823.

[2] KA Barlett, RA Scantlebury e PT Wilkinson. Uma observação sobre transmissão full duplex confiável em links half duplex, *Comm. ACM* 12 (5) (1969) 260-261.

[3) K. Birman e T. Joseph, Comunicações reabilitadas na presença de falhas, *ACM Trans. Comput. ciência* 5 (1) tl987) 47-76.

[4] K. Birman e T. Joseph, Explorando a replicação em sistemas distribuídos, em: S. Mullender, ed., *Distributed Systems* (ACM, Nova York, 1990).

[5] C. Fidge, Timestamps em sistemas de troca de mensagens que preservam a ordenação parcial, em: *Prac. IIth Australian Computer Science Conf.,* University of Queensland, 1988.

[6] JM Helary, C. Jard, N. Plouzeau e M. Raynal, Detecção de propriedades estáveis em aplicações distribuídas, em: *Prac. 6º ACM Simp. em PODC* (1987) 125-136.

[7J JM Helary e M. Raynal, *Sincronização e Controle de Sistemas e Programas Distribuídos* (Wiley, Nova York, 1990), p. 200.

[8] T. Joseph e K. Birman, gerenciamento de baixo custo de dados replicados em sistemas distribuídos tolerantes a falhas, *ACM Trans. Comyul. ciência 4* (1) (1986) 54-70.

[9] L. Lamport, Tempo, relógios e ordenação de eventos em um sistema distribuído, *Comm. ACM* 21 (7) (l97Sj 5g-ssh.

[10] F. Assuntos, Algoritmos para detecção de terminação distribuída, *Computação distribuída.* 2 (3) (1987) 161-175.

{11] F. Mattem, Tempo e locais globais de sistemas distribuídos, em: *Proc. Internet. Workshop on Parallel and Distributed* A / gorifnms, Bonas, France, 1988 (North-Holland, Amsterdam) 215-226.

{12] G. Neiger e S. Toueg, Substituindo conhecimento *comum* e em tempo real em sistemas distribuídos, em: *Proc. 6º* ***ACM*** *Simp. em PODC* (1987) 281-293.

[13J M. Raynal, *3 lgorithm for Mutual Exclusion* (MiT Press, Cambridge, MA, 1986), p. 107.

[14] M. Raynal, *Disrributed Computation and Metwor-ks: Concepts , Tools and Algorithms* (MIT Press, Cambriclge, MA, 1958), p. 166.

[15] A. Schiper, J. Eggii e A. Sandoz, Um novo algoritmo para implementar a ordenação causal, em: **f•**ror. *3º Interno. Workshop on Distributed Algorithms,* Nice, **Lecttare** Notes in Computer Science **392** (Springer, Berlin, l9fi9) 2I9—232.

[16] W. Stunning, Um protocolo de transferência de dados, *Computer Networks* 1 (1976) 99-1t0.